Wouter Bulten

← Back to category

Kalman filters explained: Removing noise from RSSI signals

Wouter Bulten on 11 Oct 2015, keywords: *javascript*, *Kalman filter*, *RSSI*, *noise*

If you have heard about *iBeacons* or indoor localization before, then you have probably also heard about *RSSI*: the **Received Signal Strength Indicator**. The RSSI value resembles the power of a received radio signal (measured in *dBm*). The higher the RSSI value, the higher the signal strength. The rationale behind using RSSI values is that almost all wireless systems report and use this value natively; i.e. no additional sensors are required to measure RSSI values. It can therefore be considered as a free input to a system.

Small Mac OS tip: Try clicking on your WiFi icon in the navbar while holding alt (or option in Mac OS terms). You will see the current RSSI of your connection (only when you are connected to a network!).

So what can we do with RSSI? Well, there is a relation between RSSI and distance (see also Figure 1). As you can probably imagine: the larger the distance between you and the sender of a signal, the lower the signal strength will be. While many researchers question the usability of RSSI measurements in general[1], I've used them extensively (and with success) for indoor localization purposes. In this article I will show you how to use RSSI measurements and, maybe even more important, to remove noise from the raw data using Kalman filters.

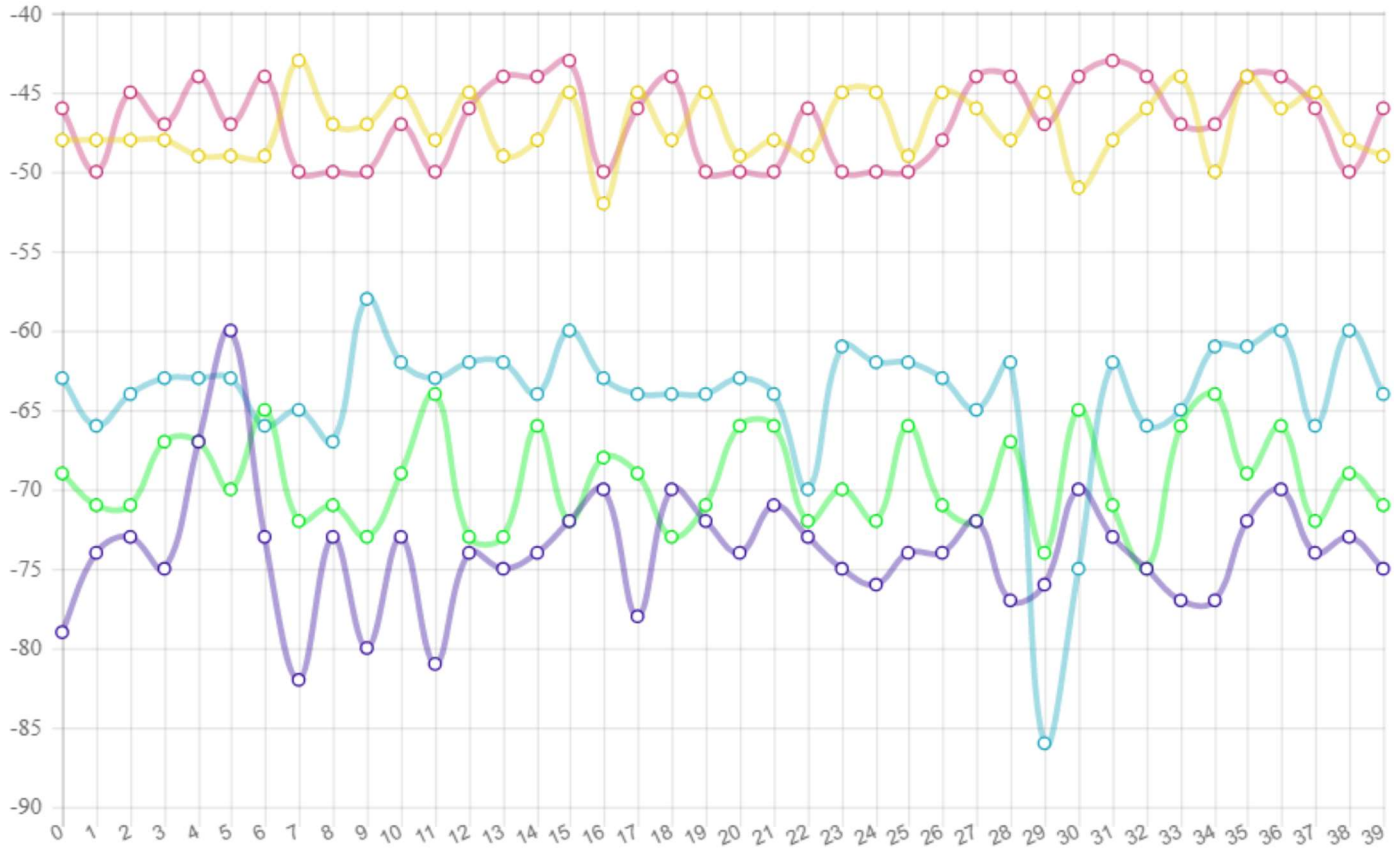


Figure 1: RSSI measurements over time. The received signal strength of a device is clearly influenced by distance but the amount of noise is substantial. For this plot, a bluetooth device was set up as a iBeacon to continuously broadcast its unique identifier. Another bluetooth device was placed at various distances from the beacon and acted as a recording device. With a 1Hz sample rate RSSI values were sampled. For the 'room' case, the beacon was placed in an adjacent room to show the effect of walls.

From RSSI to meters

RSSI is measured in *dBm* but is in its raw form not really useful in an application (apart from being a diagnostic measurement). The nice thing about RSSI is that we can translate the measurements to distance estimates in meters. More precise, we can describe the relation between RSSI and distance using a model, the *Log-Distance pathloss model*[2]:

$$\text{RSSI} = -10n \log_{10}(\frac{d}{d_0}) + A_0 \quad (1)$$

In this model, d describes the distance between the transceiver and recipient, n the signal propagation exponent and A_0 a referenced RSSI value at d_0. Usually d_0 is taken to be 1 such that A_0 becomes the signal strength measured at a distance of 1 meter of the device.

A_0 can usually be retrieved from the device itself as it is often part of the broadcast message. Alternatively, you can approximate A_0 by collecting data and averaging the signal strength measurements. The signal propagation exponent n is a constant that differs from environment to environment. For indoor applications, n is often set to 2.

1m 3m Different room

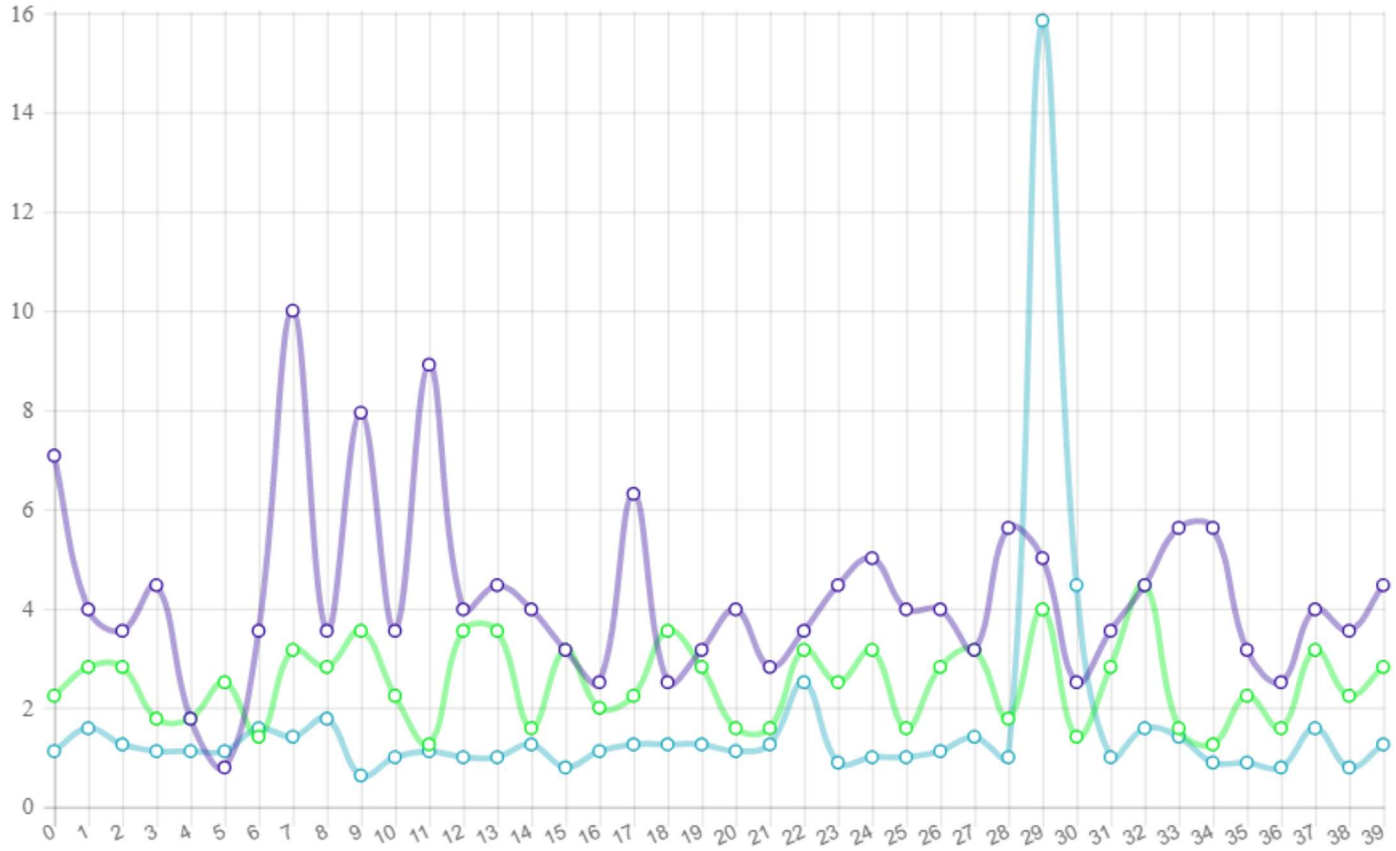


Figure 2: RSSI converted to distance in meters using the Log-distance pathloss model. Source of the data is the same as in Figure 1. Note that the distance estimations are roughly correct but contains a lot of noise.

Fighting noise

Okay, we can translate RSSI to distance, but our estimates are still not very good. As shown in Figure 2, at some point in time (roughly around t=29) our RSSI-based estimate states that the device should be 16 meters away, when in fact it is at a 1 meter distance.

Where does this discrepancy come from? Following our model, in an ideal world, the RSSI value is only dependent on the distance between the two devices. In reality, however, RSSI values are heavily influenced by the environment and have, consequently, high levels of noise. This noise is, for example, caused by multi-path reflections: signals bounce against objects in the environment such as walls and furniture.

So, **we need to fight the noise**. Here *Kalman filters* come in to play.

The Kalman filter is a state estimator that makes an estimate of some unobserved variable based on noisy measurements. It is a recursive algorithm as it takes the history of measurements into account. In our case we want to know the true RSSI based on our measurements. The regular[3] Kalman filter assumes linear models. That is, the step from the current state to the next state, and the translation from state to measurement should be linear transformations.

The general form of the transition model is as follows:

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t \tag{2}$$

The current state $\boldsymbol{x_t}$ is defined as a combination of the previous state $\boldsymbol{x_{t-1}}$ (given some transformation matrix $\boldsymbol{A}$), a control input $\boldsymbol{u}$ (this could be a movement command in a robot) and noise $\boldsymbol{\epsilon}$. THe noise is called the *process noise*: noise caused by the system itself. *Don't worry if this looks daunting, we will simplify this formula shortly. Hold on!*

For our RSSI filtering application we assume that a device doesn't move. Moreover, we assume that in the time frame of our measurement our own position is also static. In other words: over time we expect a constant RSSI signal, everything else is noise. To apply this to our model we completely ignore $\boldsymbol{u}$ and set $\boldsymbol{A}$ to an identity matrix. These two changes result in a very simple model:

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t \approx x_{t-1} + \epsilon_t \tag{3}$$

The second part of implementing the Kalman filter is defining the observation model: how does a particular state $\boldsymbol{x}$ result in a measurement $\boldsymbol{z}$? The general model is as follows:

$$z_t = C_t x_t + \delta_t \tag{4}$$

Here C is the transformation matrix and δ measurement noise (noise caused by faulty measurements). We model RSSI directly; i.e. our state and measurements are equal. This results in the following reduced measurement model:

$$z_t = C_t x_t + \delta_t \approx x_t + \delta_t \tag{5}$$

Updating the filter

With our two transitions defined we can define the prediction step of the Kalman filter. This step describes what we expect our state to be without using any measurements. As we assume a static system our prediction step is straightforward:

$$\bar{\mu}_t = \mu_{t-1} \tag{6}$$

$$\bar{\Sigma}_t = \Sigma_{t-1} + R_t \tag{7}$$

Note the difference between x and μ: μ describes our prediction and x the true value of the state. The bar above the μ denotes that we yet have to incorporate information from the measurement. Σ defines the certainty of our prediction. We base this on the previous certainty (if we were unsure about the previous measurement, the next one will be unsure too) and the process noise R which describes noise caused by the system itself. For the RSSI example we use a low value for the process noise (e.g. 0.008); we assume that most of the noise is caused by the measurements.

Using our prediction estimate we compute the Kalman gain. For our static system we have a simplified version of the normal Kalman gain:

$$K_t = \bar{\Sigma}_t (\bar{\Sigma}_t Q_t)^{-1} \tag{8}$$

The gain is used as a weighting function between the certainty of our estimate and the certainty of the measurement (influenced by the measurement noise Q). Q is set to a value that relates to the noise in the actual measurements (e.g. the variance of the RSSI signal).

When we are very uncertain about our prediction of the system (i.e. Σ is large) we should trust the measurement more. Likewise, if the measurement noise is low (i.e. Q is low) it is also wise to use those measurements. In these cases the Kalman gain will be high. At the other side, if we are very certain about our prediction of the system and/or the measurement noise is low the Kalman gain will be low. This translates directly to the update step:

$$\mu_t = \bar{\mu}_t + K_t(z_t - \bar{\mu}_t) \tag{9}$$

$$\Sigma_t = \bar{\Sigma}_t - (K_t\bar{\Sigma}_t) \tag{10}$$

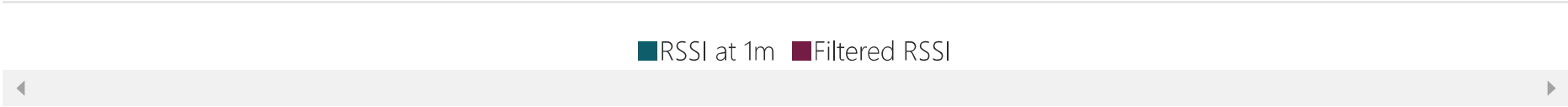
In the update step we compute the final prediction of the system (μ, without the bar) and the certainty Σ. The larger the Kalman gain the more we integrate the measurement into our state estimate. If the Kalman gain is low we trust our prediction more and only use little information of the measurement.

Try to imagine how this characteristic results in a filter: On every step, the Kalman filter decides how much of the measurement it takes in to account based on the certainty of the measurements and our state.

Putting the filter into practice

We defined our model, we defined the filter. Now it is time to put the filter to a test. Does our Kalman fitler remove the noise from the raw RSSI signal?

I applied a simple Kalman filter to the "1m" data of the RSSI example dataset. I am using my KalmanJS library which you can find on GitHub or read more about in a separate post on this blog. The results are shown in the figure below:



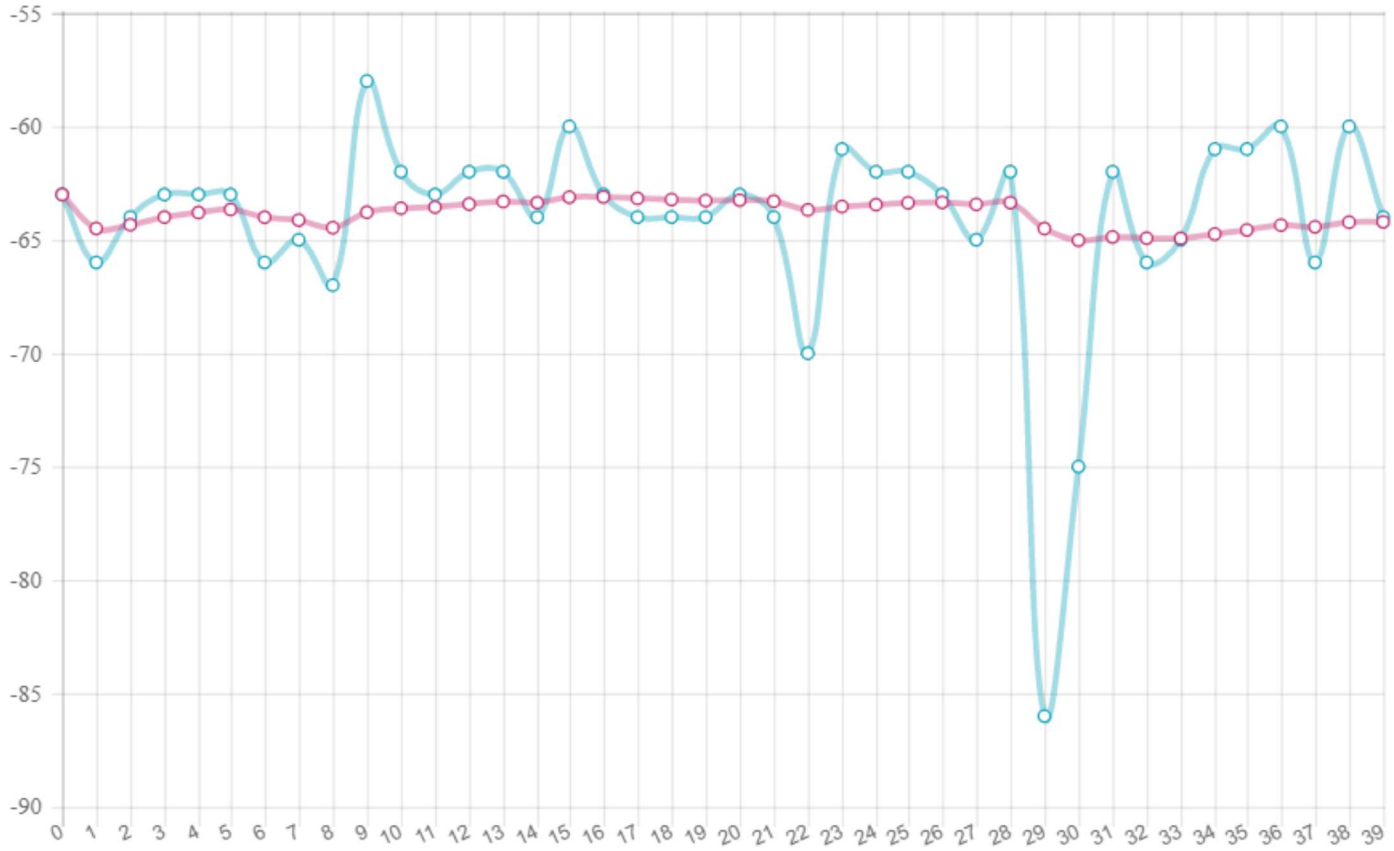


Figure 3: The effect of a Kalman filter on raw RSSI data sampled from a static device (i.e. no movement at both the receiver and transmitter end). The Kalman filter removes a large part of the noise from the signal.

By using a Kalman filter we are able to remove noise from a very noisy signal. As the update functions are easy to compute the time complexity of the filter is very low; this results in a high performing system. The only drawback is that we lose a bit of responsiveness, but get a clearer signal in return!

Any questions or comments? Please see the comments section.

Reference

This project/post was part of my research on indoor localization. Please see my paper or this presentation for more information. You can use the following reference if you want to cite my paper:

W. Bulten, A. C. V. Rossum and W. F. G. Haselager, "Human SLAM, Indoor Localisation of Devices and Users," 2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI), Berlin, 2016, pp. 211-222. doi: 10.1109/IoTDI.2015.19 URL

Or, if you prefer in BibTeX format:

```
@INPROCEEDINGS{7471364,
author={W. Bulten and A. C. V. Rossum and W. F. G. Haselager},
booktitle={2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)},
title={Human SLAM, Indoor Localisation of Devices and Users},
year={2016},
pages={211-222},
keywords={RSSI;data privacy;indoor environment;ubiquitous computing;FastSLAM;RSSI update;SLAC algorithm;device RSSI;device indoor localisation;device location;device position;environment noise;human SLAM;nontrivial environment;received signal strength indicator;simultaneous localisation and configuration;smart space;user indoor localisation;user motion data;user privacy;Estimation;Performance evaluation;Privacy;Simultaneous localization and mapping;Privacy;Simultaneous localization and mapping;Smart Homes;Ubiquitous computing;Wireless sensor networks},
doi={10.1109/IoTDI.2015.19},
month={April},}
```

References & Notes

1. Many researchers question whether RSSI values contains any value at all. For example:
   Dong, Q., & Dargie, W. (2012). *Evaluation of the reliability of RSSI for indoor localization*. In *2012 International Conference on Wireless Communications in Underground and Confined Areas, ICWCUCA 2012*. doi:10.1109/ICWCUCA.2012.6402492 ↩

2. Oguejiofor, O., Okorogu, V., Abe, A., & BO, O. (2013). *Outdoor Localization System Using RSSI Measurement of Wireless Sensor Network. International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, 2(2), 1–6. ↩

3. There is also a extended version of the Kalman filter that works with non-linear models. ↩

Other posts related to this

Lightweight Javascript library for Noise filtering using Kalman filters

Filtering noisy measurements can be an extremely difficult endeavor; take brain signals for example. While this is true, there are also many situations where the system is fairly simple and the focus lies on speed, online computation (i.e. in real time) and ease of use. Filtering distance measurements from a... Start reading →

Human SLAM, Indoor localization using particle filters

A key problem (or challenge) within smart spaces is indoor localization: making estimates of users' whereabouts. Without such information, systems are unable to react on the presence of users or, sometimes even more important, their absence. This can range from simply turning the lights on when someone enters a room... Start reading →

Website status monitor using Jenkins

Jenkins is a great tool for continuous integration and deployment. In a sense, and greatly simplified, it is a very complex and feature rich task runner. I

already use it for testing development applications (e.g. with Mocha, PHPUnit, etc.) but was wondering whether it could also keep track of applications... Start reading →

Made with Jekyll | RSS Feed